This document tries to lay out anything Gentoo-specific that you need to know in order to use bcfg2. Mostly that has to do with getting it to cooperate with the various pieces of Portage. Services, all things POSIX, and just about anything else that bcfg2 does will work the same on Gentoo as on any other distribution. bcfg2 is new on Gentoo; please let the list know if you find errors or omissions.

Prerequisites

Installing bcfg2

Early in July 2008, bcfg2 was added to the Gentoo portage tree. So far it's only keyworded for ~x86, but we hope to see it soon in the amd64 and x64-solaris ports. If you're using Gentoo on some other architecture, it should still work provided that you have a reasonably up to date Python; try adding `app-admin/bcfg2 ~*` to your `/etc/portage/package.keywords` file.

If you don?t use emerge to install bcfg2, you?ll want to make sure you have all the prerequisites installed first. For a server, you?ll need:

- `app-admin/gamin` or `app-admin/fam`;
- `dev-python/pyopenssl`;
- `dev-python/lxml`

Clients will need at least:

- `app-portage/gentoolkit`;
- `dev-python/lxml` or `dev-python/elementtree` (if you?re running python-2.4 or below);

Package Repository

You?ll need (to make) at least one archive of binary packages. The Portage driver calls `emerge` with the `?getbinpkgonly` option. See `make.conf(5)` and `emerge(1)` manpages, specifically the `PORTAGE_BINHOST` environment variable.

Time Saver: quickpkg

If you have a standing Gentoo machine that you want to preserve or propagate, you can generate a complete package archive based on the present state of the system by using the quickpkg utility. For example:

```
# for pkg in `equery -q l` ; do quickpkg "=$pkg" ; done
```

?will leave you with a complete archive of all the packages on your system in `/usr/portage/packages/All`, which you can then move to your ftp server.

Cataloging Packages In Your Repository

Once you have a set of packages, you?ll need to create a catalog for them in /var/lib/bcfg2/Pkgmgr. Here?s a template:

```
<PackageList uri='' type='portage' priority=''>
  <Group name=''>
```

```
    <Package name='' version=''/>
  </Group>
</PackageList>
```

?and a partially filled-out example, for our local Gentoo/VMware build:

```
<PackageList uri='ftp://filthy.uchicago.edu/200701-vmware/' type='portage' priority='0'>
  <Group name='gentoo-200701-vmware'>
    <Package name='app-admin/bcfg2' version='0.9.1_pre1'/>
    [...]
    <Package name='x11-wm/twm' version='1.0.1'/>
  </Group>
</PackageList>
```

The `<Group>` name (in our example, ?gentoo-200701-vmware?) should be included by any host which will draw its packages from this list. Our collection of packages for this class of machines is at the listed URI, and we only have one collection of packages for this batch of machines so in our case the `priority` doesn?t really matter, we?ve set it to `0`.

Notice that package name fields are in `CAT/TITLE` format.

Here?s a hack which will generate a list of Package lines from a system?s database of installed packages, especially useful in conjunction with the `quickpkg` example above:

```
#!/bin/bash
for pkg in `equery -q l` ; do
   title=`echo $pkg | sed -e 's/\(.*\)-\([0-9].*\)/\1/'`
   version=`echo $pkg | sed -e 's/\(.*\)-\([0-9].*\)/\2/'`
   echo "    <Package name='${title}' version='${version}'/>"
done
```

Configuring Client Machines

Set up `/etc/bcfg2.conf` the way you would for any other bcfg2 client.

In `make.conf`, set `PORTAGE_BINHOST` to point to the URI of your package repository. You may want to create versions of `make.conf` for each package repository you maintain, with appropriate `PORTAGE_BINHOST` URI's in each, and associated with that package archive's group under `Cfg` -- for example, we have `Cfg/etc/make.conf/make.conf.G99_gentoo-200701-vmware`. If a client host switches groups, and the new group needs a different set of packages, everything should just fall into place.

Pitfalls

Package Verification Issues

As of this writing (2007/01/31), we?re aware of a number of packages marked stable in the Gentoo x86 tree which, for one reason or another, consistently fail to verify cleanly under `equery check`. In some cases (pam, openldap), files which don?t (ever) exist on the system are nonetheless recorded in the package database; in some (python, bcfg2, ahem), whole classes of files (.pyc and .pyo files) consistently fail their md5sum checks; and in others, the problem appears to be a discrepancy in the way that symlinks are created vs. the way they?re recorded in the database. For example, in the OpenSSH package, /usr/bin/slogin is a symlink to ./ssh, but equery expects it to point to an unadorned ssh. An analogous situation exists with their manpages, leading to noise like this:

```
# equery check openssh
[ Checking net-misc/openssh-4.5_p1 ]
!!! /etc/ssh/sshd_config has incorrect md5sum
!!! /usr/bin/slogin does not point to ssh
!!! /usr/share/man/man1/slogin.1.gz does not point to ssh.1.gz
!!! /etc/ssh/ssh_config has incorrect md5sum
 * 62 out of 66 files good
```

We can ignore the lines for `ssh_config` and `sshd_config`; those will be caught by bcfg2 as registered config files and handled appropriately.

Because bcfg2 relies on the client system?s native package reporting tool to judge the state of installed packages, complaints like these about trivial or intractable verification failures can trigger unnecessary bundle reinstalls when the bcfg2 client runs. bcfg2 will catch on after a pass or two that the situation isn?t getting any better with repeated package installs, stop trying, and list those packages as ?bad? in the client system?s statistics.

Aside from filing bugs with the Gentoo package maintainers, your narrator has been unable to come up with a good approach to this. Maybe write a series of `Rules` definitions according to what the package database thinks it should find, and/or stage copies of affected files under `Cfg`, and associate those rules and files with the affected package in a bundle? Annoying but possibly necessary if you want your stats file to look good.

/boot

Gentoo as well as some other distros recommend leaving `/boot` unmounted during normal runtime. This can lead to trouble during verification and package installation, for example when `/boot/grub/grub.conf` turns up missing. The simplest way around this might just be to ensure that `/boot` is mounted whenever you run bcfg2, possibly wrapping bcfg2 in a script for the purpose. I?ve also thought about adding `Action` clauses to bundles for grub and our kernel packages, which would mount `/boot` before the bundle installs and unmount it afterward, but this doesn?t get around the problem of those packages flunking verification.